NASA CR-159142

FORTRAN

Subroutines for Out-of-Core Solutions of Large Complex Linear Systems

Elizabeth L. Yip

Boeing Commercial Airplane Company
Seattle, Washington

N/\S/\

National Aeronautics and
Space Administration

November 1979

CONTENTS

1.0 SUMMARY

This document describes the design and usage of two main subprograms using direct methods to solve large linear complex systems of the form $Ax = b$, whose coefficient matrices are too large to be stored in core. In the first main subprobram, the basic idea is to reduce the matrix to an upper triangular matrix and subsequently solve the problem by backward substitution. A row interchanging strategy called *"threshold pivoting"* is adopted to preserve numerical stability and to minimize disk-transfers. This algorithm does *not* yield the usual LU factorization of some row permutation of the coefficient matrix. The second main subprogram is designed for linear systems with a certain sparse structure, namely, the matrix can be written as $B + D$ where B is a block-banded matrix, and D has only a few columns of nonzeros. A variant of the Sherman-Morrison updating formula is employed in this case. The second main subprogram calls the first main subprogram to solve $B(x,y) = (b,u)$ where x, y are unknowns, b is the right-hand side of the linear system $(B + D)x = b$, and U is the nonzero columns of D.

2.0 INTRODUCTION

This document describes the design and usage of two FORTRAN main subprograms for the CDC digital computer systems, namely subroutine ETCGP and subroutine ETCSM. These subroutines were written as part of a research effort investigating unsteady transonic flow. References 1 through 5 present a procedure for analyzing the flow about harmonically oscillating airfoils and wings in the transonic regime. This procedure is based on small perturbations. A solution is formulated using finite difference techniques which results in a large set of simultaneous equations, written matrix form as

$$Ax = b \tag{1}$$

where A is a sparse complex matrix of order equal to the number of mesh points. The matrix b is a complex matrix with the number of columns equal to the number of modes for which pressure distributions are to be found. The matrix A may be of order 3,500 for a practical two-dimensional airfoil.

Numerical solutions to equation (1) were initially obtained using relaxation techniques. However, for a significant range of practical values of Mach number and reduced frequency, experience showed that relaxation techniques applied to equation (1) failed to converge. The matrix A in equation (1) does not possess any of well-known properties (e.g., positive definiteness, diagonal dominance) which guarantee the convergence of relaxation methods. However, the physical origin of equation (1) guarantees the existence of a unique solution. Thus the obvious alternative is a direct solution method, which assumes no properties of the matrix A other than existence of a unique solution for an equation of the form of (1) when any arbitray matrix b is applied.

Out-of-core algorithms have been discussed in standard textbooks of numerical analysis (e.g. Reference 6). These algorithms are numerically as stable as the well-known regular Gaussian elimination, yet they are not designed for efficient execution in modern day computers. The physical origin of equation (1) makes "blocking" of the matrix A an obvious and attractive data structure. Yet existing "blocked linear equation solvers" at best assume non-singularity and the well-conditioning of the i-th diagonal sub-block at the i-th blocked pivotal step (e.g. Reference 7). This property is unnatural and is not based on the physics of the problem. Out basic algorithm does not require the matrix A to have this particular property. As stated in the above paragraph, the only property of A it assumes is non-singularity of the matrix A. It also takes into consideration numerical stability and computational efficiency. Sections 43 and 44 discuss these advantages in detail.

The FORTRAN subroutines described here are applicable to both dense and sparse matrices that are too large to be stored in core. Thus they are useful for solving any equation of the form of equation (1), in particular in the case when there is no useful properties of the matrix A to guarantee convergence of any iterative methods. For example, subroutine ETCGP (as well as the version for real matrices) has been used in different research and production computer codes at the Boeing Company.

2

Results of applying the routines of this document together with th program described in reference 5 are presented in reference 4. The development of the routines of this document was in conjunction with work described in references 4, 5, and 8.

The first step in developing the algorithm for solving equation (1) is to partition the augmented matrix [A|B] into blocks so that it can be considered to have the following structure:

$$
\begin{bmatrix}
A_{11} & A_{12} \cdots A_{1S} & B_1 \\
A_{21} & A_{22} \cdots A_{2S} & B_2 \\
\cdot & & \\
\cdot & & \\
\cdot & & \\
A_{S1} & A_{SS} & B_S
\end{bmatrix}
$$

Figure 1. - Data Structure

Each block is stored as a record in mass storage with the requirement that at least three blocks can be held in core simultaneously, and the diagonal blocks are square blocks. The mass storage used should be a random access file (called direct access file in the IBM nomenclature).

The methods used in both main subprograms are direct methods. In the first main subprogram, the basic idea is to reduce the coefficient matrix to an upper triangular system. In the second main subprogram, the coefficient matrix is assumed to have the for B + D, where B is a block-banded matrix, and D is a matrix with only a few columns of nonzeros. A variant of the Sherman-Morrison updating formula is employed.

Features of the two subprograms include options to
- Solve more than one set of right-hand sides
- Control the frequency of pivoting
- Access the submoduli of the main subprograms
- Take advantage of the block sparsity of the coefficient matrix.

In section 3.0 we shall list our symbols and nomenclatures. In section 4.0, we shall discuss the algorithms in detail, analyze their numerical stabilities, remark on their FORTRAM implementation and compare our approaches with outer approaches outlined in the literature. In secion 5.0, we shall discuss in detail the usage of the two subprograms.

3.0 NOMENCLATURE

$A = (a_{i,j})$ A is a matrix, the entry of the i-th row and j-th column is denoted by $a_{i,j}$

$A = (A_{i,j})$ A is a matrix partitioned into blocks, the block in the i-th block row and the j-th block column is denoted by $A_{i,j}$

A^T The transpose of A

Upper triangular matrix   Matrix with all zeros below the diagonal

Block profile matrix   Matrix of the form:

(where the shaded area indicates non-zero entries)

4
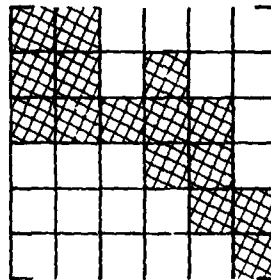
Block-Banded
matrix

Mat ix of the form:
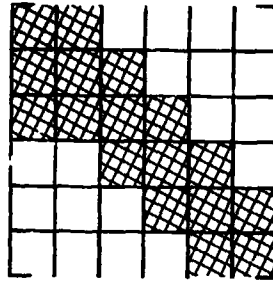
(Where the shaded area indicates non-zero entries)

Threshol. pivoting
and pivotal
tolerance

Threshold pivoting is a row interchanging strategy controlled by a pivotal tolerance parameter μ. The μ parameter is a real number such that $0 \leqslant \mu \leqslant 1$. If $A = \left(a_{i,j}\right)$, and $a_{i,i} \geqslant \max\limits_{j > i} \left|a_{i,j}\right|$, then there is no row interchanging, otherwise interchange rows i and m where $m = \max\limits_{j > i} \left|a_{ij}\right|$

Random Access file

Multi-record mass storage input/output file which allows the user to create, access and modify its records on a random basis without regard for their physical positions or internal structures. (They are called direct access files in IBM nomenclature.)

Sherman-Morrison Updating Formula for $A = B + UV^T$:

$$A^{-1} = B^{-1} - B^{-1}U\left(I_4 + V^TB^{-1}U\right)^{-1}V^TB^{-1}$$

Matrix norms: $\|A\| = \max_{i} \left\{ \sum_{j} |a_{ij}| \right\}$

$\|A\|_{\infty} = \max_{j} \left\{ \sum_{i} |a_{ij}| \right\}$

$\|A\|_{2} = \max_{i} \left\{ \left(\sum_{j} a^{2}_{ij} \right)^{1/2} \right\}$

Condition Number of a nonsingular matrix

$$K_{m}(A) = \|A\|_{m} \cdot \|A^{-1}\|_{m} \qquad\qquad m = 1, 2,....\infty$$

Growth (growth factor) of a reduction process	The largest absolute value of the numbers generated by the reduction process

4.0 DISCUSSION

In this section, we highlight the special features of our algorithms. In sections 4.1 to 4.4, we shall discuss Algorithm I, which reduces the coefficient matrix to an upper triangular matrix and solves the problem by backward substitution. Then we shall remark on FORTRAN implementation and compare the efficiency of this algorithm with the approaches outlined by Bjorck and Dahquist (ref. 6), and Reid, (private communication).

In sections 4.5 and 4.7, we shall discuss Algorithm II, which employs a variant of the Sherman-Morrison updating formula. Then we shall remark on its FORTRAN implementation, and the numerical stability of our particular application of the Sherman-Morrison updating formula.

4.1 ALGORITHM I (SUBROUTINE ETCGP)

For the convenience of discussion, we shall firstly assume the coefficient matrix A to be dense. We shall show how the algorithm can be modified for a block-profile system.

We shall first illustrate the algorithm with a 3 x 3 block system, and assume no row interchanging is necessary. In this case, the algorithm yields Crout's LU decomposition. The LU factors can replace the original coefficient matrix A on disk. Assume, graphically, A is stored on disk as in the following figure.

$$
\begin{bmatrix}
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33}
\end{bmatrix}
$$

Figure 2.—Original Matrix

In the *first* block pivotal step, we form the LU factorization of A_{11}, $A_{11} = L_1 U_1$ and replace $A_{11}, A_{12}, A_{13}, A_{21}, A_{31}$ as in figure 3.

$$\begin{bmatrix} L_1U_1 & L_1^{-1}A_{12} & L_1^{-1}A_{13} \\ A_{21}U_1^{-1} & A_{22} & A_{23} \\ A_{31}U_1^{-1} & A_{32} & A_{33} \end{bmatrix}$$

Figure 3.—After First Block Pivotal Step

For $i = 2, 3$, write $L_1^{-1}A_{1i}$ as U_{1i}, and $U_1^{-1}A_{i1}$ as L_{i1}.

In the second block pivotal step, replace A_{22} with $A_{22} - L_{21}U_{12}$, the decompose the resultant into its LU factors, and complete the rest of the second block pivotal step as illustrated in fig. 4.

$$\begin{bmatrix} L_1U_1 & U_{12} & U_{13} \\ L_{21} & A_{22} & A_{23} \\ L_{31} & A_{32} & A_{33} \end{bmatrix} \longrightarrow \begin{bmatrix} L_1U_1 & U_{12} & U_{13} \\ L_{21} & \begin{array}{c}(L_2U_2)\\ A_{22}-L_{21}U_{12}\end{array} & L_2^{-1}\left(A_{23} - L_{21}U_{13}\right) \\ L_{31} & U_2^{-1}\left(A_{32} - L_{31}U_{12}\right) & A_{33} \end{bmatrix}$$

Figure 4.—Second Block Pivotal Step

Write $L_2(A_{23}-L_{21}U_{13})$ as U_{23}, and $U_2(A_{32}-L_{31}U_{12})$ as L_{32}. In the third pivotal step, we replace A_{33} with $A33 - L_{31}U_{13} - L_{32}U_{23}$ and decomplse the resultant to its LU factors. The third block pivotal step can be illustrated by fig. 5.

$$\begin{bmatrix} L_1U_1 & U_{12} & U_{13} \\ L_{21} & L_2U_2 & U_{23} \\ L_{31} & L_{32} & A_{33} \end{bmatrix} \longrightarrow \begin{bmatrix} L_1U_1 & U_{12} & U_{13} \\ L_{21} & L_2U_2 & U_{23} \\ L_{31} & L_{32} & \begin{array}{c}\left(L_3U_3\right)\\ A_{33} - L_{31}U_{13} - L_{32}U_{23}\end{array} \end{bmatrix}$$

Figure 5.—Third Block Pivotal Step

8

Thus we have reduced the matrix A to an upper triangular matrix, and thus equation (1) can be solved by backward substitution.

Now we shall explain our row interchanging strategy, which is commonly known as *threshold pivoting*. A real number μ is chosen so the $0 \leqslant \mu \leqslant 1$. We perform row interchanging only when $|a_{ii}| < \mu \cdot \underset{j>i}{\text{Max.}} \ |a_{ji}|$.

Note that if $\mu = 0$, there is no row interchanging at all; if $\mu = 1$, we have the regular row interchanging. We shall illustrate the application of this row interchanging strategy to a 2 x 2 block system:

$$\begin{bmatrix} U_1 \\ A_{21} \end{bmatrix} = \left[\begin{array}{c|c} u_{11} & u_{12} \\ 0 & a_{22} \\ \hline a_{31} & a_{32} \\ a_{41} & a_{42} \end{array} \right]$$

Figure 6.—First Pivotal Step of a 2 x 2 Block System

In the first pivotal step, we first decompose A_{11} into its LU factors with pivoting, i.e., $A_{11} = L_1 U_1 P_1$, then replace A_{12} with $L_1^{-1} P_1 A_{12}$.

Let $U_{12} = L_1^{-1} P_1 A_{12}$. Now suppose

$$U = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

and

$$|u_{11}| > \mu \cdot \text{max.} \ \left\{ |a_{31}| \ , \ |a_{41}| \right\} = \mu \cdot |a_{31}|$$

We have to interchange row 1 of U_1 with row 1 of A_{21}, then eliminate u_{11} and a_{41} with a_{31} as the pivot, the resultant is as follows:

$$\begin{bmatrix} a_{31} & a_{32} \\ 0 & u_{22} \\ 0 & \bar{u}_{12} \\ 0 & \bar{a}_{42} \end{bmatrix}$$

9

Now suppose

$$|u_{22}| < \mu \cdot \max \left\{ |u_{12}| , |a_{42}| \right\} = \mu \cdot |a_{42}|$$

We have to interchange tow 2 with row 2 of A_{21}, and eliminate u_{12} and u_{22} with a_{42} as a pivot, the resultant is:

$$\begin{bmatrix} a_{31} & a_{32} \\ 0 & \bar{a}_{42} \\ \hline 0 & 0 \\ 0 & 0 \end{bmatrix}$$

After the first block pivotal step, the disk storage should be as follows:

$$\begin{bmatrix} L_1 U_1 & L_1 P_1 A_{12} \\ \hline \text{Multipliers and pivoting information} & A_{22} \end{bmatrix}$$

(In the formal description of our algorithm, we shall write $L_1 P_1 A_{12}$ as U_{12}, and the disk record that stores the multipliers and pivoting information as indicated in the previous diagram L_{21}.)

The following is a formal description of Algorithm I:

Algorithm I

Let A be an n x n block system

For i = 1 step 1 until n do

    For j = i step 1 until n do

        Read A_{ij} into core

            For k = 1 step 1 until i-1 do

            Read L_{jk}, U_{kj} into core

            Modify $\begin{bmatrix} U_{kj} \\ A_{ij} \end{bmatrix}$ by repeating the row operations done to $\begin{bmatrix} U_{kj} \\ A_{ki} \end{bmatrix}$.

            If there is row interchanging between $\begin{bmatrix} U_k \\ A_{ki} \end{bmatrix}$, rewrite U_{kj} on disk.

Enddo

if j=i

$$A_{ii} = L_i U_i P_i$$

Else

$$A_{ij} = L_i^{-1} P_i A_{ij}$$

Endif

Rewrite A_{ij} on disk (rename A_{ij} as U_i.

Enddo

For j = 1 step 1 until n do

   Read A_{ij} into core

   For k = 1 step 1 until i - 1 do

      Read L_{jk}, U_{ki} into core

      Modify $\begin{bmatrix} U_{ki} \\ A_{ji} \end{bmatrix}$ by repeating the row operations done to $\begin{bmatrix} U_k \\ A_{jk} \end{bmatrix}$.

      If there is row interchanging between $\begin{bmatrix} U_k \\ A_{jk} \end{bmatrix}$,rewrite U_{ki} on disk.

   Enddo

   Eliminate A_{ji} from $\begin{bmatrix} U_i \\ A_{ji} \end{bmatrix}$ with row interchanging if necessary.

   Store multipliers and pivoting information as L_{ji} on disk.

   Enddo

Enddo

Repeat the same row operation to the right-hand side, and solve for x by backward substitution

Note :

1. It t ie is no interblock pivoting, it requires two disk read/write's in the innermost do-loop; otherwise, it requires three.

2. Nowhere in the algorithm do we assume non-zeros in the diagonal blocks. Although nonzeros in the final upper triangular form will indicate matrix singularity, and backward substitution will be impossible to carry out.

Now we shall show how the algorithm can b▪ modified for a block-profile system.
Define an n x n matrix K such that:

K(i. j)= 0 indicates the (i, j)block is a zero block

K(i. j)≠ 0 indicates K(i, j)is the location of the (i, j)block (a nonzero block) on disk.

For example, for the following block-profile matrix,
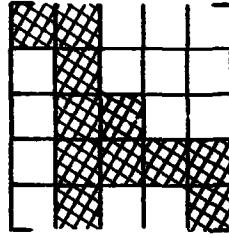
Where the shaded area indicates the nonzero blocks, its corresponding K matrix can be

$$
\begin{bmatrix}
1 & 2 & & & \\
3 & & & & \\
4 & 5 & & & \\
6 & 7 & 8 & 9 & \\
10 & & & 11 &
\end{bmatrix}
$$

We call the K matrix the profile matrix of the matrix A.

In subroutine ETCGP, we scan the profile matrix to determine the beginning and end of each block row and block column, and change the "ends" the do-loops in the algorithm accordingly. To handle the "fill-in's"of the zero blocks (i.e., if the (i, j) block is originally a zero block. but the reduction process turns it into a nonzero block), we find the maximum entry of the original K matrix, m, and set K(i, j)= m + 1, update m by adding 1 to it, and revise the information on the beginnings and ends of the i-th block row and j-th block column if necessary.

4.2 REMARKS ON FORTRAN IMPLEMENTATION OF ALGORITHM I

1. Storage of the multipliers and pivoting information

    Suppose the i, j block is a nonzero block that is $n_i \times n_j$, and i>j. The number of words in its corresponding record on disk is $2*(n_i*n_j)+n_j+1$ Before the j-th block pivotal step. the first $2*(n_i*n_j)$ words stored the entries of the (i,j) block, and the last n_j+1 words are zeros. After the j-th block pivotal step, the first $2*(n_i*n_j)$ words store the multipliers used. and the last n_j+1 words store the pivoting information. We can consider this record as consist-

ing of two separate arrays: an array of complex numbers, $A(n_j, n_j)$, and an array of integers, $I(n_j+1)$. In the 2 x 2 block system we used to illustrate the threshold pivoting discussion, after the first pivotal step, the (2,1) block of fig 6., that is A_{21}, is stored as an array of complex numbers, $A(2,2)$, as

$$\begin{bmatrix} U_{11}/a_{31} \\ a_{41}/a_{31} \\ U_{12}/a_{42} \\ U_{22}/a_{42} \end{bmatrix}$$

and the pivotal information is stored in the integer array, $I(3)$, as

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

The two arrays are packed as one record on disk.

The matrix A stores the actual multipliers used in the reduction process. $I(1)=1$ indicates the first row of A_{21} interchanged with row 1 of the upper triangular matrix U_1 before the first column of A_{21} is eliminated. $I(2)=2$ indicates the second row of A_{21} interchanged with row 2 of U_1 before the second column of A_{21} is eliminated. $I(3)=I(1)+I(2)>0$ indicates there is interblock pivoting between the first block row (the pivotal block row) and the second block row. If there is no interblock pivoting in this step, $I(1)=0$, and $I(2)=0$, thus $I(3)=0$. As it is obvious from this example, $I(k)=0$ indicates that no row interchange is needed to eliminate the k-th column of the (i,j) block. $I(k)=m>0$ indicates that the m-th row of the (i,j) block is interchanged with the k-th row of the corresponding upper triangular matrix U_j before the k-th column of the (i, j) block is eliminated, and $I(n_j+1)=I(1)+I(2)+...+I(n_j)$. If $I(n_j+1)=0$, it indicates that there is no interblock pivoting between the i-th and j-th rows; if $I(n_j+1)>0$, it indicates that there is interblock pivoting between the i-th and j-th block row.

2. Optimization of the Innermost Loop

As indicated by the 3 x 3 block system in section 4.1, if there is no interblock pivoting, the most expensive operation is the matrix operation

$$A_{ij} \longleftarrow A_{ij} - L_{ik} \cdot U_{kj}$$

In subroutine ETCGP, this operation is carried out by a Compass subroutine CCSMAB which is about five times faster than straight FORTRAN. Even if there is interblock row pivoting, we still can take advantage of CCSMAB. Consider the situation in the innermost do-loop in the description of Algorithm 1. We have the following blocks in core:

$$\begin{bmatrix} 0 & U_{kj} \\ L_{ik} & A_{ij} \end{bmatrix}$$

If there is no pivoting between the i-th and j-th block row, the operation can be represented by the following equation:

$$L \begin{bmatrix} U_{kj} \\ A_{ij} \end{bmatrix} = \begin{bmatrix} U_{kj} \\ \hat{A}_{ij} \end{bmatrix}$$

where L is such that

$$L^{-1} = \begin{bmatrix} I & 0 \\ L_{ik} & I \end{bmatrix}$$

If there is pivoting between the i-th and j-th block rows, then the operation can be represented by the following equation:

$$\hat{L}P \begin{bmatrix} U_{kj} \\ A_{ij} \end{bmatrix} = \begin{bmatrix} \hat{U}_{kj} \\ \hat{A}_{ij} \end{bmatrix}$$

where P is that permutation matrix that interchanges the rows and L is such that

$$\hat{L}^{-1} = I + P \begin{bmatrix} 0 & 0 \\ L_{ik} & I \end{bmatrix}$$

We can write

$$\hat{L}^{-1} = \begin{bmatrix} L_0 & 0 \\ M & I \end{bmatrix}$$

Therefore

$$\hat{L} = \begin{bmatrix} L_0^{-1} & 0 \\ -ML_0^{-1} & I \end{bmatrix}$$

Also write

$$P \begin{bmatrix} U_{kj} \\ A_{ij} \end{bmatrix} = \begin{bmatrix} \overline{U}_{kj} \\ \overline{A}_{ij} \end{bmatrix}$$

Therefore

$$\hat{L} \begin{bmatrix} \overline{U}_{kj} \\ \overline{A}_{ij} \end{bmatrix} = \begin{bmatrix} L_0^{-1} \overline{U}_{kj} \\ \overline{A}_{ij} - ML_0^{-1} \overline{U}_{ik} \end{bmatrix}$$

Thus in ETCGP, we compute $\hat{U}_{kj} = L_0^{-1} \overline{U}_{kj}$ then use CCSMAB to compute $A_{ij} - M\hat{U}_{kj}$.

Note that M is just the lower block of $P \begin{bmatrix} 0 \\ L_{ik} \end{bmatrix}$ and can be computed quite efficiently by

forward substitution. (The following describe the compution M and $U = L_{ij}^{-1} U_{kj}$.
M would overwrite L_{ik}, and U would over write U_{kj}. We shall call L_{ik}. M and U_{kj}. U and
the actual row and column dimension of U, NR, and NC.)

For m=2 step 1 until NR do
    If l(m)>0
    k=l(m)
    For h=1 step 1 until NC do

$$U'(m, h) = U(m, h) - \sum_{s=1}^{m-1} M(k, s) * U(s, j)$$

    Enddo

15

For h=1 step 1 until m-1 do

    M(k,h)=0

Enddo

Else

    (no operation)

Endif

Enddo

3. Pivotal Tolerance

At this point, we cannot give any theoretical guidanc: on the choice of the pivotal tolerance μ. Experimaentally, $\mu = 0.001$ proves to be satisfactory. In ETCGP, we store μ in a labelled common block

    COMMON/ETPIVOT/U

and U is set to 0.001 by at DATA statement. The user can reset the value of U by any executable statement before calling ETCGP.

4. Monitoring of Stability

ETCGP keeps track of the *"growth"* of the reduction process (which is the largest absolute value of all the numbers generated by the reductic 1 process). Our motivation for keeping track of the growth is explained in section 4.3.

ETCGP also generates an extra right-hand side row which is the row sum of the coefficient matrix A. It subtracts (1.0,0) from each element in the solution of t⊦ ₃ extra right-hand side. The resultant gives some indication of the *"smallness"* of the residue Ax-b of the actual problem.

4.3 NUMERICAL STABILITY OF ALGORITHM I

Reid (ref. 9) and Wilkinson (ref. 10) have analyzed the stability of the ₁ ₅ular Gaussian elimination. If L and U are respectively computed lower and up⌐₊₁ triangular factors of A, the A, L and U are related asfollows:

    $A = LU + E$

with $E=(e_{i, j})$ and $|e_{i, j}| \leqslant 3.01 * m * \epsilon * g$

where m is the order of A, and ϵ is the machine pr._ision, (:n CDC equip:.ient, $\epsilon = 10^{-14}$), g is the growth of the reduction process.

Reid and Wilkinson analysis cannot be extended in any obvious manner to give a practical bound for E for Algorithm 1 of this document. The best we can do at present is the following:

$$A = M_1^{-1}M_2^{-1} \ldots M_n^{-1}U + M_1^{-1}M_2^{-1} \ldots M_n^{-1}E_n + \ldots + M_1^{-1}M_2^{-1}E_2 + M^{-1}E_1$$

where M_i is the matrix that performs the ith block pivotal step

$$E_i = \left(e_{kl}{}^i\right) \qquad \left|e_{kl}{}^i\right| \leqslant 3.01 * \epsilon * g_i \quad \text{with } g_i \text{ being the growth of}$$

the first ith block pivotal step.

The main difficulty is due to the fact that we do not use the same pivots for eliminating the subdiagonal elements in the same column. Thus

$$M_1^{-1}M_2^{-1} \ldots M_i^{-1}E_i \neq E_i \qquad \text{for} \qquad i = 1, 2, \ldots, M$$

whilst equality holds for the above expression in the regular Gaussian elimination. However each block pivotal step the process of eliminating the lower triangular elements of $A_{i, i}$ from

$A_{i, i}$	$A_{i, i+1}$	$\cdots \quad \cdots$	$A_{i, n}$

and the process of elimination $A_{j, i}$ from

$U_{i, i}$	$U_{i, i+1}$	$\cdots \cdots \cdots$	$U_{i, n}$
$A_{j, i}$	$A_{j, i+1}$	$\cdots \cdots \cdots$	$A_{j, n}$

for $i \leqslant j \leqslant n$ are regular Gaussian elimination. Thus the *"growth"* of Algorithm 1 at least gives the local stability of these substeps. The norm of the difference of the computed solution from the actual solution give a realistic bound for the norm of E, because the actual solution x satisfies the equation Ax = b, the computed solution y satisfies the equation (A + E)y =b.

Thus

$$Ax = (A + E)y$$

$$A(\quad) = Ey$$

$$\|Ey\| \leqslant \|A_2\| \|x - y\|$$

In all our test problems

$$\|x - y\| \geq 10^{-11}$$

4.4 COMPARISON OF ALGORITHM I WITH OTHER APPROACHES

Bjorck and Dahlquist (ref. 6) and Reid (ref. 8) have proposed approaches to the problem of solutions of large dense linear systems. Bjorck and Dahlquist's approach follows.

The matrix is partitioned into block rows:

$$
\begin{array}{|c|}
\hline
A_1 \\
\hline
A_2 \\
\hline
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\cdot \\
\hline
A_n \\
\hline
\end{array}
$$
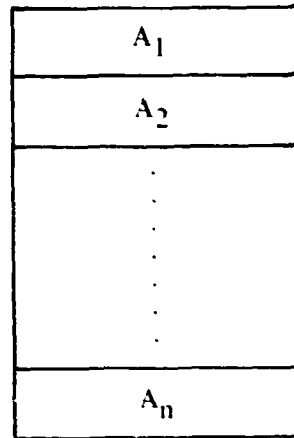
with the requirement that a minimum of two block rows can be held in core simultaneously.

The first block pivotal step can be described as follows:

1. Read A_1 into core and reduce it to $\boxed{\overline{A_1}}$ with regular Gaussian elimination.

2. For $j=2,3,\ldots,n$ read A_j into core reduce

$$
\boxed{\overline{A_1}} \qquad \boxed{A_1}
$$

to

$$
\boxed{A_j} \qquad \boxed{A_j}
$$
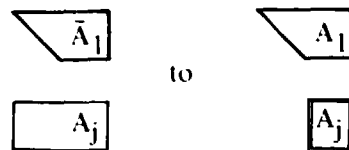
with row-interchanges when necessary, and write A_j back onto disk.

3. Write A_1 back onto disk.

The other pivotal steps are similar.

Reid proposed the same data structure as we do. His first block pivotal substeps consists of the following substeps to be performed for $i=2,3,\ldots,n$.

1. Read into core the blocks $\begin{pmatrix} A_{11} \\ A_{i1} \end{pmatrix}$ and apply normal Gaussian elimination, overwriting

eliminated elements by multipliers and using row interchanges. Note that for $i > 3$ the modified block A_{11} is upper triangular and advantage may be taken of this.

2. For $j=2,3...n$ read in blocks $\begin{pmatrix} A_{1j} \\ A_{ij} \end{pmatrix}$ modify them using the multipliers and interchanges held in $\begin{pmatrix} A_{11} \\ A_{i1} \end{pmatrix}$ and then write them back to disk.

3. Write modified block A_{i1} to disk.

The remaining block pivotal steps are performed similarly. Operations on the right-hand side vector may be performed subsequently using the stored multipliers or at the same time as the elimination.

Reid has proved that given the same amount of core, the block row storage scheme is only efficient for matrices of order of less than 300. For large matrices, Algorithm I requires the least amount of disk input/output. The following table summarizes the features of the three approaches.

Approaches	Features	No. of disk access
Dahlquist and Bjorck	two block rows are needed in core	$4\dfrac{m^4}{N^2}$
Reid	four blocks are needed in core four disk read/write's in the inner most do-loop	$\dfrac{32}{3}\left(\dfrac{m}{\sqrt{N}}\right)^3$
Algorithm I (without pivoting)	three blocks are needed in core t·· ɔ disk read/write's in the inn most do-loop	$\dfrac{2}{3}\left(\dfrac{m}{\sqrt{N}}\right)^3$
Algorithm I (with the worst possible case of pivoting)	three disk read/write in the inner most do-loop	$\dfrac{5\sqrt{3}}{2}\left(\dfrac{m}{\sqrt{N}}\right)^3$

* m is the order of the matrix, and N is one half of the number of words available in core.

19

4.5 ALGORITHM II (SUBROUTINE ETCSM)

Algorithm II is designed for coefficient matrix with a special sparsity structure:

$$A = \begin{bmatrix}
x\ x & & & & & & & & \\
x\ x\ x & & & & & & & & \\
& x\ x\ x & & & & & & & \\
& & x\ x\ x & & & & & & \\
& & x\ x\ x\ x & & & & & \\
& x & & x\ x\ x & & & & \\
& x & & x\ x\ x\ x & & & \\
& x & x & & x\ x\ x & & \\
& x & x & & & x\ x\ x \\
& x & x & & & & x\ x
\end{bmatrix}^{*}$$

We can write A = B + D where B is a banded matrix:

$$B = \begin{bmatrix}
x\ x & & & & & & & & \\
x\ x\ x & & & & & & & \\
& x\ x\ x & & & & & & \\
& & x\ x\ x & & & & & \\
& & & x\ x\ x & & & & \\
& & & & x\ x\ x & & & \\
& & & & & x\ x\ x & & \\
& & & & & & x\ x\ x & \\
& & & & & & & x\ x
\end{bmatrix}$$

and D = A -B:

$$D = \begin{bmatrix}
x & \\
x & \\
x & x \\
x & x \\
x & x \\
x & x
\end{bmatrix}$$

*The x's are single elements

20

D can be written as $D = UV^T$, where U and V^T are as follows:

$$U = \begin{bmatrix} o & o \\ o & o \\ o & o \\ o & o \\ x & o \\ x & x \\ x & x \\ x & x \\ x & x \\ x & x \end{bmatrix} \qquad V^T = \begin{bmatrix} o & o & 1 & o & o & o & c & \jmath & o & o \\ o & o & o & o & 1 & o & o & o & o & o \end{bmatrix}$$

Thus to solve a $x = b$, we apply the Sherman-Morrison updating formula

$$x = A^{-1}b = \left(B + UV^T\right)^{-1}b$$

$$= B^{-1}b - B^{-1}u\left(I + V^TB^{-1}U\right)^{-1}V^TB^{-1}b$$

The algorithm II can be described as follows:

1. Reduce B to an upper triangular matrix with Algorithm I.
2. Repeat the row operation in 1 to [b|U]. Solve for B [z,y] = [b,U] by backward substitution.
3. Compute [S,T] = V^T [z,y].
4. Decompose (I+T) into its LU factors with row interchanging.
5. Solve for w in the equation (I-T) w=S by forward and substitution.
6. Compute x=S-T*x.

4.6 REMARKS ON FORTRAN IMPLEMENTATION OF ALGORITHM II

In this section, we shall use the notation we have defined so far.

1. Storage of U and V

We partitioned U comformable to the partitioning of the coefficient matrix A:

$$U = \begin{Bmatrix} U_1 \\ U_2 \\ . \\ . \\ . \\ . \\ U_n \end{Bmatrix}$$

and store them on disk in the same block which stores the corresponding partition of the right-hand side.

$$\begin{bmatrix} b_1 & U_1 \\ b_2 & U_2 \\ . & . \\ . & . \\ . & . \\ b_n & U_n \end{bmatrix}$$

Since the matrix V only consists of ones and zeros, we do not store V explicitly.

We use two-dimensional array:

    INTEGER N(MR,MC)

where MR $\geqslant$ the number of block columns that contain element columns in the matrix $D=UV^T$, and MC $\geqslant$ 2 + the total number of columns in the matrix U. The contents of N are defined as follows:

Let $i_1, i_2, \ldots$ be the indices of the blocks that contains columns of D, and then

$\quad$ N(1,1)=i_1

$\quad$ N(2,1)=i_2

$\quad\quad$.

$\quad\quad$.

$\quad\quad$.

Suppose for j=1,2. . . all (i_j,m) blocks for some $m > k_j$ contain the nonzero elements of D, then

$$N(1,2)=k_1$$
$$N(2,2)=k_2$$

.

.

.

Suppose for j=1,2 . . . , D contains the following columns of the i_j-th block:

$m_{j,1}, m_{j,2}, \ldots, m_{j,n(j)}$, with $m_{j,1} < m_{j,2} < \cdots < m_{j,n(j)}$.

Then

$$N(1,3)=m_{1,1}, \ N(1,4)=m_{1,2}, \ldots, N(1,n(1)+2)=m_{1,n(1)}, \ N(1,n(1)+3)=0$$
$$N(2,3)=m_{2,1}, \ N(2,4)=m_{2,2}, \ldots, N(2,n(2)+2)=m_{2,n(2)}, \ N(2,n(1)+3)=0$$

.

.

.

It is important that $N(j,n(j)+3)$ be set to zero, so that the program know the $m_{j,n(j)}$ is the last column from the i_j-th block column to go into D. The following 3 x 3 block system will illustrate our scheme:

$$A = \begin{bmatrix}
X\ X & & & & & & & \\
X\ X\ X & & & & & & & \\
\ \ X\ X\ X & & & & & & & \\
\ \ \ \ X\ X\ X & & & & & & & \\
\ \ \ \ X\ X\ X & & & & & & & \\
\hline
X\ \ \ \ X & X\ X & & & & & \\
X\ \ \ \ X & X\ X\ X & & & & & \\
X\ \ \ \ X & \ \ X\ X\ X & & & & & \\
X\ \ \ \ X & \ \ \ \ X\ X\ X & & & & & \\
X\ \ \ \ X & \ \ \ \ \ \ X\ X & & & & & \\
\hline
X\ \ \ \ X & & X\ X & & & \\
X\ \ \ \ X & & X\ X\ X & & & \\
X\ \ \ \ X & & \ \ X\ X\ X & & & \\
X\ \ \ \ X & & \ \ \ \ X\ X\ X & & & \\
X\ \ \ \ X & & \ \ \ \ \ \ X\ X & & &
\end{bmatrix}$$

$$D = \begin{bmatrix}
\begin{array}{cc|c|c}
 & & & \\
 & & & \\
\hline
x & x & & \\
x & x & & \\
x & x & & \\
x & x & & \\
x & x & x & \\
\hline
x & x & x & \\
x & x & x & \\
x & x & x & \\
x & x & x & \\
x & x & x & \\
\end{array}
\end{bmatrix}$$

N(1,1)=1 meaning the first block column contains columns for D

N(2,1)=2 meaning the second block column contains columns for D

N(1,2)=2 meaning in the first block column, only (m,1),m ≥ 2 contains columns for D

N(2,2)=3 meaning in the second block column, only (m,2),m ≥3, contains columns for D

N(1,2)=3,N(1,4)=5,N(1,5)=0 meaning only the third and fifth columns of the first block
  column belongs to D

N(2,3)=3,N(2,4)=0 meaning only the third column of the second block column belongs
  to D

2. Computation of $V^T(z,y)$ (Step 3 of Algorithm II)

Since V is not stored explicityly, we use the following:

(Let NB be the number of block columns that contain column of D, L be the total number
of nonzero columns of D, and the two-dimensional array N, is as defined as before, and
NC=L+number of right-hand sides.)

g=1

For ι=1 step 1 until NB do

    k=N(i,1)

    Read T_k=b_k,U_k into core

    For j=3 step 1 until LU+2 do

        m=N(i, j)

        If m=0, then exit do-loop with index j

Otherwise

$$S(s,h)=T_k(\text{in},h)$$

Enddo

$$g=g+1$$

Enddo

Note that the array S contains $v^T(z,y)$

4.7 NUMERICAL STABILITY OF ALGORITHM II

Suppose A and B are matrices of order n and are related in the following manner:

$$A = B + UV^T \tag{2}$$

v here U and V are nxr matrices with $r \ll n$, and both U and V are of rank r. Then it follows from a variant of the well known Sherman-Morrison formula (ref. 6) that

$$A^{-1} = B^{-1} - B^{-1}U(I_r + V^T B^{-1}U)^{-1}V^T B^{-1} \tag{3}$$

where I_r is the identity matrix of order r. Assume B is well-conditioned and has been decomposed into LU factors B=LU, then equation (3) provides a very efficient method to solve the linear system

$$Ax = b. \tag{4}$$

However, a general concern when using equation (3) is that the matrix $\left(I_r + V^T R^{-1}U\right)$ may be ill-conditioned. In this section, we prove a sufficient condition for the well conditioning of $\left(I_r + V^T B^{-1}C\right)$

Before we proceed, we state the usually accepted definition of the condition number of a matrix and an inequality related to it.

Let A be an nxm, $n \geq m$, matrix with linearly independent columns, then the conditional number of A, denoted by k(A) is such that

$$k(A) = \frac{\max_{\|x\|=1} \|Ax\|}{\min_{\|x\|=1} \|Ax\|}$$

It is also true that

$$k(A) \geqslant \|A\|_2 \cdot \|A^+\|_2 \tag{5}$$

where $A^+ = (A^T A)^{-1} A^T$, the generalized inverse of A.

Let R(A) be the range of A. Inequality (5) is obvious from the fact that:

$$\min_{\|x\|_2 \neq 1} \|Ax\|_2 = \min_{\|x\| \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\| \neq 0} \frac{\|x\|}{\|Ax\|} = \frac{\|x_0\|}{\|Ax_0\|} = \frac{\|A^+ A x_0\|}{\|Ax_0\|} \leqslant \max_{\|y\| = 1} \|A^+ y\|$$

Further, if A is nonsingular, equality will hold.

We now prove the following:

Theorem 1.   If A and B are related as in equation (2), and A and B are invertible, then

$$\left(I - V^T B^{-1} U\right) = U^+ A B^{-1} U \tag{6}$$

$$\left(I - V^T B^{-1} U\right) = V^T B^{-1} A \left(V^T\right)^+ \tag{7}$$

where $U^+ = \left(U^T U\right)^{-1} U^T$, and $\left(V^T\right)^+ = V \left(V^T V\right)^{-1}$ are the generalized inverses of U and V, respectively.

Proof: Note

$$A = B - U V^T$$

$$= \left(I_n - U V^T B^{-1}\right) B$$

$$A B^{-1} = \left(I_n - U V^T B^{-1}\right)$$

$$\therefore A B^{-1} U = \left(U - U V^T B^{-1} U\right)$$

$$\therefore A B^{-1} U = \left(U_r - V^T B^{-1} U\right)$$

$$\therefore \left(I_r - V^T B^{-1} U\right) = U^+ A B^{-1} U$$

Thus we have proved the validity of equation (6). The proof of equation (7) is similar

26

The following equations are direct derivations from equations (6) and (7):

$$\left(I - V^T B^{-1} U\right)^{-1} = U + B A^{-1} U \qquad (8)$$

$$\left(I - V^T B^{-1} U\right)^{-1} = V^T A^{-1} B \left(V^T\right)^{+} \qquad (9)$$

Let $\ell_1 = \left(\|U\|_2 \cdot \|U^+\|_2\right)^2$, $\ell_2 = \left(\| V^T\|_2 \cdot \|\left(V^T\right)^+\|_2\right)^2$, $\ell_3 = \|U\|_2 \cdot \|U^+\|_2 \cdot \|V^T\| \cdot \|\left(V^T\right)^+\|_2$

Combining equations ((6), (7), (8), and (9) with the fact that $\|CD\|_2 \leqslant \|C\|_2 \|D\|_2$
for any two matrices C and D,

$$k\left(I - V^T B^{-1} U\right) = \left\| I - V^T B^{-1} U\right\|_2 \left\|\left(I - V^T B^{-1} U\right)^{-1}\right\|_2$$

$$k\left(I - V^T B^{-1} U\right) \leqslant \min. \ \{\ell_1, \ell_2, \ell_3\} \ k(A) \ k(B). \qquad (10)$$

Inequality (10) states that if A and B are well-conditioned, and either U or V is well-conditioned
then $I + V^T B^{-1} U$ is well-conditioned.

Inequality (10) is useful because $\|U_2\|$, $\|U^+\|_2$, $\|V^T\|$ and $\|\left(V T\right)^+\|$ can be
computed quite economically and the physical problems that yield matrices A and B usually
give some indication of their well-conditioning. The well-conditioning of B and $\left(I - V^T B^{-1} U\right)$
ensures that the solutions of $Bx_0 = b$

$$By = U$$

$$\left(I + V^T B^{-1} U\right)z = V^T x_0$$

can be computed with satisfactory accuracy. Thus the solution of Ax=b via equation (3)
can be computed with satisfactory accuracy (11).

In our particular application of the Sherman-Morrison updating formula, $V^T V$ is a permu-
tation matrix, thus $\|v^+\|_2 \cdot \|v\|_2 = 1$. Therefore as long as A and B are both well-condi-
tioned, Algorithm II is *always stable*.

27

5.0 COMPUTER PROGRAM USAGE

5.1 MACHINE AND EXECUTION ENVIRONMENT

The algorithms described in the previous section are programmed as a FORTRAN subroutine library (which we call OCSLIB out-of-core solution library). The subroutines are written for the FTN compiler of the CDC computers.

5.2 OPERATING SYSTEM

This subroutine library is designed for NOS 1.1. Its compass subroutines are optimized for the CDC 6600.

5.3 TIMING

Timing is hardware and operating system dependent. The following formula gives a very rough estimate for the timing:

$$CP \text{ second} = 1/2 * n * p^2 * k$$

where n is the order of the linear system, and p is the band-width and k is machine-dependent. To estimate k, make a sample run and compute k using the above formula. For the Cyber 175, $k \cong 8 \times 10^{-5}$.

5.4 FILES AND FILE FORMAT

OCSLIB uses at least one random access file. OCSLIB has each block of the coefficient matrix as a record. If the block is $n_i \times n_j$, then the record length is $2*n_i*n_j + n_j + 1$.

5.5 USAGE

5.5.1 USING ALGORITHM I (SUBROUTINE ETCGP)

We recommend the following procedure:

Step 1.   Define the block system of the coefficient matrix: choose a sequence of positive

integers $n_1, n_2, \ldots n_N$ to partition the matrix into a block system like fig. 1 of section 1.0, so that the block $A_{i,j}$ is $n_i \times n_j$. The partition is to be chosen so that at least three blocks can be held in core simultaneously.

Step 2.   Define the block profile of the matrix: define an array

INTERGER INDEX (M,M+1)

such that $M \geqslant N$, and INDEX(i, j)=0 if the $A_{i,j}$ is a zero block.

INDEX(i,j)$\geqslant$0 if the $A_{i,j}$ has at least one nonzero entries.

(here we consider $b_k = A_{i,N+1}$, for k=1,2, ... N)

Step 3.   Write the augumented matrix on disk for ETCGP,n_k (k=1,2, . . . N) at a time. In this step we provide a subroutine ETBMGEN to write a block row of the augmented matrix on disk in a format that is acceptable by ETCGP. Thus we shall describe the usage and argument list of ETBMGEN.

COMPLEX W(MXR,MXC)

INTEGER INDEX(MT,MT+1),JN(NT2),NDBLK(NT1) (where NT1$\geqslant$NTBLKS+1

NT2$\geqslant$NTBLKS**2 + NTBLKS+3)

.

.

DO 10 I=1, NTBLKS

(Code to generate the i-th block row of the augmented matrix.)

CALL ETBMGEN(I,NTAPE,IN,MT,NTBLKS,WORK,MXR,MXC)

10   CONTINUE

The argument list for ETBMGEN is described as follows:

```
'UL- OUTINE ETBMGEN(I,NTAPE,INDEX,MT,JN,NTBLKS,NDBLK,WORK,
   S4XR,MXC)
C***************\*******************************************************
C
C      PURPOS.    TO WRITE THE NON-ZERO BLOCKS OF A BLOCK ROW OF MATRIX
C                 IN ETCGP FORMAT
C
C      ARGUMEN. LIST
C
C            I - BLOCK ROW INDEX
C            NTAPE - OUTPUT DISK FILE
C                    OUTPUT - ARGMENTED MATRIX IN ETCGP FORMAT
C            INDEX - 2 DIMENSIONAL ARRAY FOR THE PROFILE OF THE MATRIX
C.                   ROW DIMENSION = MT, COLUMN DIMENSION = MT+1
C.                   INDEX(I,J)=0  (I,J)-TH BLOCK IS ZERO
C                    INDEX(I,J).GT.0  LOCATION OF (I,J)-TH BLOCK ON NTAPE
C            'IT - ROW DIMENSION OF THE ARRAY INDEX
C            JN - INTEGER ARRAY FOR THE RANDOM ACCESS FILE NTAPE, AT LEAST
C                 (NTBLKS+NTBLKS*NTBLKS+3) MANY WORD'S LONG
C            NTBLKS - NO. OF BLOCK ROWS
C            NDBLK - ARRAY TO STORE BLOCK SIZES,NDBLK(NTBLKS+1)=NO.OF RHS
C            WORK - (2 DIMENSICNAL  TO USER)
C                   INPUT ARRAY FOR THE NON-ZERO BLOCKS OF THE I-TH BLOCK ROW
C            MXR - ROW DIMENSION OF WORK
C            MXC - COLUMN DIMENSION OF WORK
C
C      NOTE    SUBROUTINE ETCGP ASSUMES ALL THE NON-ZEROBLOCKS TO BE
C              DENSE BI'CKS.  (SEE THE SAMPLE CALLING PROGRAM)
C
C*********************************************************************
```

ETBMGEN alters the contents of the array INDEX.

Step 4.   To solve the linear system via ETCGP

COMPLEX WORK (LW)

INTEGER INDEX(MT,MT,+1),NDBLK(NT1)

.

.

.

CALL ETCGP(NTAPE,INDEX,NTBLKS,NDBLK,WORK,LW,ITAG)

The calling sequence of ETCGP is as follows:

SUBROUTINE ETCGP(NTAPE,INDEX,NDBLKS,NDBLK, WORK,LWORK,ITAG)

```
      SUBROUTINE ETCGP(NTAPE,INDEX,NTBLKS,NDBLK,WORK,LWORK,ITAG)
      OPTION,KEEP=OFF,INLIST=ON
C*****************************************************************************
C
C     PURPOSE  MAIN SUBROUTINE
C            REDUCE MATRIX TO AN UPPER TRIAGULAR MATRIX - CALL ETCGPRM
C            REPEAT ROW OPERATION ON RHS - CALL ETCGPFS
C            BACKWARD SUBSTITUTION - CALL ETCGPBS
C
C     ARGUMENT LIST
C            NTAPE - INPUT/OUTPUT DISK FILES FOR MATRIX
C                      INPUT - ORIGINAL ARGMENTED MATRIX
C                      OUTPUT - ROW OPERATIONS PERFORMED AND UPPER
C                               TRIANGULAR FORM AND SOLUTION
C            INDEX - 2 DIMENSIONAL ARRAY FOR THE PROFILE OF THE MATRIX
C                      ROW DIMENSION = NTBLKS, COLUMN DIMENSION = NTBLKS+1
C                      INDEX(I,J)=0  (I,J)-TH BLOCK IS ZERO
C                      INDEX(I,J).GT.0  LOCATION OF (I,J)-TH BLOCK ON NTAPE
C            NTBLKS - NO. OF BLOCK ROWS
C            NDBLK - ARRAY TO STORE BLOCK SIZES,NDBLK(NTBLKS+1)=NO.OF RHS
C            WORK - WORKING ARRAY (COMPLEX TO USER)
C            LWORK - LENGTH OF THE ARRAY WORK REGARDED AS COMPLEX
C                      LWORK.GE. (NTBLKS**2 + 3*NTBLKS + 3*MXB + 3*N)
C                      WHERE MXB = (MXBLK**2) + (MXBLK+2)/2 WITH MXBLK = THE
C                      MAXIMUM OF ALL THE ENTRIES OF THE ARRAY NDBLK, AND WHERE
C                      N IS THE ORDER OF THE SYSTEM
C            ITAG - COMPUTATIONAL PATH
C                      ITAG=1 REDUCE MATRIX TO UPPER TRIANGULAR FORM AND
C                               SOLVE AX=B
C                      ITAG=2 REDUCE MATRIX ONLY
C                      ITAG=3 SOLVE AX=B ASSUMING A HAS BEEN REDUCED
C
C*****************************************************************************
```

(The array WORK should be equivalenced to the arrays W and JN as follows:

EQUIVALENCE (WORK,W) (WW(1,MXC+1),JN)

Step 5.   Read the solution from NTAPE. In this subroutine we provide a subroutine ETRDSOL which has exactly the same calling sequence as ETBMGEN. Its usage is described below:

DO 10 I=1,NTBLKS

CALL ETRDSOL(NTAPE,INDEX,MT,NTBLK,W,MXR,MXC)

(code to process the part of the solution corresponding to the i-th block row)

10 CONTINUE

5.5.2 USING ALGORITHM II (SUBROUTINE ETCSM)

We recommend the following procedure:

Step 1.   Same as step 1 of section 5.5.1.

Step 2.   Same as step 2 of section 5.5.1, except call subroutine ETSMGEN
           instead of ETBMGEN. The argument list of ETSMGEN is as follows:

```
      SUBROUTINE ETSMGEN(I,NTAPE,INDEX,MT,JN,NTBLKS,NDBLK,WORK,
     $              MXR,MXC,NB,MLB,LB,LU)
C
C **********************************************************************
C
C      PURPOSE   WRITE THE MATRIX IN A FORM ACCEPTABLE BY ETCSM
C
C      ARGUMENT LIST
C            I,NTAPE,INDEX,MT,JN,NTBLKS,NDBLK,WORK,MXR,MXC - SEE SUBROUTINE
C              ETBMGEN
C            NB,MLB,LB,LU GIVE THE SPARSITY STRUCTURE OF THE MATRIX D
C            NB - DIMENSIONED AS
C                    INTEGER NB(MLB,LU+2)
C                    NB(I,1)=K, THE K-TH BLOCK ROW CONTAINS THE COLUMNS OF D
C                    NB(I,2)=J, ONLY THE BLOCKS (M,K),M.GE.J CONTAINS THE COLUMNS
C                    OF C
C                    FOR T.GE.3, NB(I,T)=S, THE S-COLUMNS OF THE K-TH BLOCK COLUMN
C                    IS IN D.  IF D ONLY CONTAINS H COLUMNS OF THE K-TH BLOCK COLUMN
C                    THEN SET NB(I,H+2)=0
C            MLB - ROW DIMENSION OF NB
C            LB - TOTAL NO. OF BLOCK COLUMNS THAT CONTAINS COLUMNS OF D
C            LU - TOTAL NO. OF NON-ZERO COLUMNS OF D
C
C **********************************************************************
```

Step 3.   Solve the linear system via ETCSM. The argument list of ETCSM is as follows:

```
      SUBROUTINE ETCSM(NTAPE,INDEX,NTBLKS,NDBLK,WORK,LWORK,ITAG,
     $KNB,MLB,LB,LU)
C**********************************************************************
C
C      PURPOSE   APPLY THE SHERMAN-MORRISON UPDATING FORMULA TO SOLVE
C                (BE+D)X=B, WHERE BE IS A BLOCKED BENDED MATRIX,
C                AND D ONLY CONSISTS OF A FEW COLUMNS OF NON-ZEROES.
C                THE NON-ZERO COLUMNS OF D IS ASSUMED TO BE STORED WITH THE RHS.
C
C      ARGUMENT LIST
C            NTAPE,INDEX,NTBLKS,NDBLK,WORK,LWORK - SEE SUBROUTINE ETCGP
C            ITAG - THE VALUE OF ITAG IS PASSED ON TO ETCGP
C            KNB,MLB,LB,LU GIVE THE SPARSITY STRUCTURE OF THE MATRIX D
C            KNB - DIMENSIONED AS
C                    INTEGER KNB(MLB,LU+2)
C                    KNB(I,1)=K, THE K-TH BLOCK ROW CONTAINS THE COLUMNS OF D
C                    KNB(I,2)=J, ONLY THE BLOCKS (M,K),M.GE.J CONTAINS THE COLUMNS
C                    OF D
C                    FOR T.GE.3, KNB(I,T)=S, THE S-COLUMNS OF THE K-TH BLOCK COLUMN
C                    IS IN D.  IF D ONLY CONTAINS H COLUMNS OF THE K-TH BLOCK COLUMN
C                    THEN SET KNB(I,H+2)=0
C            MLB - ROW DIMENSION OF KNB
C            LB - TOTAL NO. OF BLOCK COLUMNS THAT CONTAINS COLUMNS OF D
C            LU - TOTAL NO. OF NON-ZERO COLUMNS OF D
C
C**********************************************************************
```

5.5.3 USING THE ITERATIVE REFINEMENT SUBROUTINE (ETCIT)

ETCIT requires the original matrix and the output matrix from TECGP to be in different files. The usage of ETCIT is obvious from the explanation of its argument list:

```
      SUBROUTINE ETCIT(NTAPEI,INDEXI,NTAPEO,INDEXO,NDBLK,NTBLKS,
     $WORK,LWORK)
C**********************************************************************
C
C      PURPOSE   ITERATIVE REFINEMENT
C
C      ARGUMENT LIST
C           NTAPEI - INPUT DISK FILE STORES ORIGINAL MATRIX
C           INDEXI - 2 DIMENSIONAL ARRAY FOR THE PROFILE OF THE MATRIX IN
C                    NTAPEI (SEE EXPLANATION FOR THE ARGUMENT INDEX IN
C                    SUBROUTINE ETCGP)
C           NTAPEO - OUTPUT DISK FILE FROM ETCGP, STORES THE UPPER TRIANGULAR FORM
C                    AND THE ROW OPERATIONS PERFORMED BY ETCGP, ALSO THE
C                    SOLUTION FROM ETCGP.
C                    THE FINAL SOLUTION FROM ETCIT WILL OVERWRITES THE SOLUTION
C                    FROM ETCGP.
C           INDEXO - 2 DIMENSIONAL ARRAY FOR THE PROFILE OF THE MATRIX IN
C                    NTAPEO (SEE EXPLANATION FOR THE ARGUMENT INDEX IN
C                    SUBROUTINE ETCGP)
C           NTBLKS - NO. OF BLOCK ROWS
C           NDBLK  - ARRAY TO STORE BLOCK SIZES,NDBLK(NTBLKS+1)=NO.OF RHS
C           WORK   - WORKING ARRAY (COMPLEX TO USER)
C           LWORK  - LENGTH OF THE ARRAY WORK REGARDED AS COMPLEX
C                    LWORK.GE. (NTBLKS**2 + 3*NTBLKS + 3*MXB + 3*N)
C                    WHERE MXB = (MXBLK**2) + (MXBLK+2)/2 WITH MXBLK = THE
C                    MAXIMUM OF ALL THE ENTRIES OF THE ARRAY NDBLK, AND WHERE
C                    N IS THE ORDER OF THE SYSTEM
C
C**********************************************************************
```

5.6 ERROR MESSAGES

5.6.1 ERROR MESSAGES FROM ETCGP

1. WORKING SPACE TOO SMALL, AT LEAST ************* WORDS ARE NEEDED RETURN FROM ETCGP.

2. WRONG CHOICE OF COMPUTATION PATH, ITAG SHOULD EQUAL TO 1,2 OR 3.

3. MATRIX SEEMED SINGULAR, EXIT FROM ETCGP.

5.6.2 ERROR MESSAGES FROM ETCSM

1. WORKING SPACE TOO SMALL, AT LEAST ************* WORDS ARE NEEDED RETURN FROM ETCSM.

2. MATRIX SEEMED SINGULAR, EXIT FROM ETCSM.

5.6.3 ERROR MESSAGES FROM ETCIT

1. WORKING SPACE TOO SMALL, AT LEAST ************* WORDS ARE NEEDED RETURN FROM ETCIT.

2. CONVERGENCE TOO SLOW, RETURN FROM ETCIT.

5.7 SAMPLE PROBLEMS

5.7.1 SAMPLE PROBLEM 1

```
      PROGRAM TGP(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,TAPE8
C
C
C     THIS PROGRAM ILLUSTRATES THE USE OF ETCGP.
C     THE PROFILE OF THE MTRIX IS BLOCK TRIADIAGONAL.
C     THE SUBBLOCKS ARE 2 X 2 BLOCKS.
C     WE CALL THE RANDOM NUMBER GENERATOR TO GENERATE THE MATRIX ENTRIE
C     WE USE THE ROW SUM OF THE MATRIX AS OUR RHS.
C
C
      COMPLEX W(1000),WW(10,100)
      INTEGER IN(10,11),NDBLK(11),JN(115),KN(110),NC(10)
C
C     NOTE THE W ARRAY SHOULD BE EQUIVALENCED TO WW
C     AND JN SHOULD BE EQUIVALENCED TO WW(1,MXC+1)
      EQUIVALENCE (WW,W),(KN,IN),(JN,WW(1,12))
      DATA KN/110*0/,W/1000*(0.,0.)/
      DATA NTBLKS/5/,NDBLK/2,2,2,2,2,1/
C
C     NOTE THE W ARRAY SHOULD BE EQUIVALENCED TO WW
C     AND JN SHOULD BE EQUIVALENCED TO WW(1,MXC+1)
      EQUIVALENCE (WW,W),(KN,IN),(JN,WW(1,12))
      DATA
C
C     GENERATE INDEX ARRAY IN IN
C
      NT1=NTBLKS
      NT1=NTLBKS+1
      NTM1=NTBLKS-1
      IN(1,1)=1
      IN(1,2)=2
      IN(1,NT1)=3
      NS=3
      DO3I=2,NTM1
      IN(I,I-1)=NS+1
      IN(I,I)=NS+2
      IN(I,I+1)=NS+3
      IN(I,NT1)=NS+4
      NS=NS+4
3     CONTINUE
      IN(NTBLKS,NTM1)=NS+1
      IN(NTBLKS,NTBLKS)=NS+2
      IN(NTBLKS,NT1)=NS+3
      NS=NS+3
```

```
C
C           FIND OUT HOW MANY NON-ZERO COLUMNS IN THE BLOCK ROW
C
        DO4I=1,NTBLKS
        NC(I)=0
        DO4J=1,NTBLKS
      4 IF (IN(I,J).GT.0)NC(I)=NC(I)+NDBLK(J)
C
        WRITE(6,100)((IN(I,J),J=1,11),I=1,10)
    100 FORMAT(*INDEX ARRAY IN :*,/,(11I5))
C       GENERATE NTBLKS BLOCK ROW OF THE MATRIX
C
        DO10L=1,NTBLKS
        NC1=NC(I)+1
C
C       GENERATE NC(I)*20 RANDOM NUMBERS
        CALL NOGEN(W,NC(I)*20)
C
C       ZERO OUT WW(.,NC1)FOR ROW SUM
C
        ND=NDBLK(I)
C
C       COMPUTE ROW SUM
        MC=NC(I)
C
        DO6J=1,ND
        WW(J,NC1)=(0.,0.)
        DO6K=1,MC
      6 WW(J,NC1)=WW(J,NC1)+WW(J,K)
C
C       WRITE I-TH BLOCK ROW FOR ETCGP
C
        CALL ETBMGEN (I,8,IN,10,JN,NTBLKS,NDBLK,WW,10,11)
C
     10 CONTINUE
C
        CALL ETCGP(8,IN,NTBLKS,NDBLK,W,1000,1)
C
C       TO RED SOLUTION FROM TAPE8
C
        DO11I=1,NTBLKS
        CALL ETRDSOL (I,8,W,10,JN,NTBLKS,NDBLK,WW,10,11)
        ND=NDBLK(I)
     11 WRITE(6,101)I,(WW(J,1),J=1,ND)
    101 FORMAT(1X,I5,*-TH BLOCK SOLUTION*,,/,(8E10.4))
        STOP
        END
```

5.7.2 SAMPLE PROBLEM 2ᵇ

```
      PROGRAM TSM(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,TAPE8)
C
C     THIS PROGRAM ILLUSTRATES THE USE OF ETCSM.
C     THE PROFILE OF THE MATRIX IS OF THE FORM A=B+D, WHERE B IS
C     BLOCK TRIDIANGONAL, AND D ONLY CONSISTS OF 2 NON-ZERO ROWS.
C     THE SUBBLOCKS ARE 2 X 2 BLOCKS.
C     WE CALL THE RANDOM NUMBER GENERATOR TO GENERATE THE MATRIX ENTRIES
C     WE USE THE ROW SUM OF THE MATRIX AS OUR RHS.
C
      COMPLEX W(1000),WW(10,100)
      INTEGER IN(10,11),NDBLK(11),JN(115),KN(110),NC(10),NB(2,5)
C
C     NOTE THE W ARRAY SHOULD BE EQUIVALENCED TO WW
C     AND JN SHOULD BE EQUIVALENCED TO WW(1,MXC+1)
      EQUIVALENCE (WW,W),(KN,IN),(JN,WW(1,12))
      DATA KN/110*0/,W/1000*(0.,0.)/
      DATA NTBLKS/5/,NDBLK/2,2,2,2,2,1/
C
C     GENERATE INDEX ARRAY IN IN
C
      NT1=NTBLKS
      NT1=NTBLKS+1
      NTM1= NTBLKS-1
      IN(1,1)=1
      IN(1,2)=2
      IN(1,NT1)=3
      NS=3
      DO3I=2,NTM1
      IN(I,I-1)=NS+1
      IN(I,I)=NS+2
      IN(I,I+1)=NS+3
      IN(I,NT1)=NS+4
      NS=NS+4
    3 CONTINUE
      IN(NTBLKS,NTM1)=NS 1
      IN(NTBLKS,NTBLKS)=NS+2
      IN(NTBLKS,NT1)=NS+3
      NS=NS+3
C
      DO5I=4,NTBLKS
      NS=NS+1
    5 IN(I,2)=NS
C
C     DEFINE THE STRUCTURE OF D
C
      NB(1,1)=2
      NB(1,2)=4
      NB(1,3)=1
      NB(1,4)=2
C
      NB(1,5)=0
C     FIND OUT HOW MANY NON-ZERO COLUMNS IN THE BLOCK ROW
C
```

```fortran
      DO4I=1,NTBLKS
      NC(I)=0
      DO4J=1,NTBLKS
    4 IF (IN(I,J).GT.0)NC(I)=NC(I)+NDBLK(J)
C
      WRITE(6,100)((IN(I,J),J=1,11),I=1,10)
  100 FORMAT(*INDEX ARRAY IN:*,/,(11I5))
C     GENERATE NTBLKS BLOCK ROW OF THE MATRIX
C
      DO10I=1,NTBLKS
      NC1=NC(I)+1
C
C     GENERATE NC(I)*20 RANDOM NUMBERS
      CALL NOGEN(W,NC(I)*20)
C
C     ZERO OUT WW(.,NC1) FOR ROW SUM
C
      ND=NDBLK(I)
C
C     COMPUTE ROW SUM
      MC=NC(I)
C
      DO6J=1,ND
      WW(J,NC1)=(0.,0.)
      DO6K=1,MC
    6 WW(J,NC1)=WW(J,NC1)+WW(J,K)
C
C     WRITE I-TH BLOCK ROW FOR ETCGP
C
      CALL ETSMGEN (I,8,IN,10,JN,NTBLKS,NDBLK,WW,10,11,NB,2,1,2)
C
   10 CONTINUE
C
      CALL ETCSM(8,IN,NTBLKS,NDBLK,W,1000,1,NB,2,1,2)
C
C     TO READ SOLUTION FROM TAPE8
C
      DO11I=1,NTBLKS
      CALL ETRDSOL (I,8,IN,10,JN,NTBLKS,NDBLK,WW,10,11)
      ND=NDBLK(I)
   11 WRITE(6,101)I,WW(J,1),J=1,ND)
  101 FORMAT(1X,I5,*-TH BLOCK SOLUTION*,,/,(8E10.4))
      STOP
      END
```

REFERENCES

1. Ehlers, F. Edward: *"A Finite Difference Method for the Solution of the Transonic Flow Around Harmonically Oscillating Wings,"* NASA CR-2257, January 1974.

2. Weatherill, W. H.; Ehlers, F. E.; Sebastian, J. D.: *"Computation of the Transonic Perturbation Flow Fields Around Two - and Three-Dimensional Oscillating Wing,"* NASA CR-2599, December 1075.

3. Weatherill, W. H.; Sebastian, J. D.; and Ehlers, F. E.: *"The Practical Application of a Finite Difference Method to the Analysis of Transonic Flow Over Oscillating Airfoils and Wings,"* NASA CR-2933, December 1977.

4. Weatherill, Warren H.; Ehlers, F. Edward; Yip, Elizabeth; and Sebastion, James D.: *"Further Investigation for Finite Difference Procedure for Analyzing the Transonic Flow About Harmonically Oscillating Airfoils and Wings,"* NASA CR-3195, 1979.

5. Weatherill, W. H.; and Ehlers, F. E.: *"A User's Guide for A344 - A Program Using a Finite Difference Method to Analyze Transonic Flow Over Oscillating Airfoils,"* NASA CR-159141.

6. Bjorck, A.; Dahlquist, D.: *"Numerical Methods,"* Prentice-Hall, Inc. 1974.

7. Calahan, D. A.: *"A Block-Oriented Sparse Equation Solver for the CRAY-1,"* Proceedings on 1979 International Conference on Parallel Processing. Bellaire, Mich. August 27 to 24, 1979.

8. Ehlers, F. Edward; Weatherill, Warren H.; Yip, Elizabeth; and Sebastion, James D.: *"An Investigation of Several Factors Involved in a Finite Difference Procedure for Analyzing the Transonic Flow About Harmonically Oscillating Airfoils and Wings,"* NASA CR-159143, 1979.

9. Reid, J. K., *"A Note on the Stability of Gaussian Elimination,"* J. Inst. Math Applics. (1971) pp. 374-375.

10. Wilkinson, J. H., *"The Algebraic Eigenvalue Problem,"* London, Oxford University Press., 1965.

11. Wilkinson, J. H., *"Rounding Errors in Algebraic Processes,"* Prentice-Hall, 1963.

1 Report No	2 Government Accession No	3 Recipient's Catalog No
NASA CR-159142		

4 Title and Subtitle	5 Report Date
FORTRAN Subroutines for Out-of-Core Solutions of Large Complex Linear Systems	November 1979
	6 Performing Organization Code

7 Author(s)	8 Performing Organization Report No
Elizabeth L. Yip	BCS-40283

9 Performing Organization Name and Address	10 Work Unit No
Boeing Commercial Airplane Company	
P.O. Box 707	11 Contract or Grant No
Seattle, Washington 98124	NAS1-15128
	13 Type of Report and Period Covered

12 Sponsoring Agency Name and Address	Final Report
Langley Research Center	
National Aeronautics and Space Administration	14 Sponsoring Agency Code
Washington, D.C. 20546	

15 Supplementary Notes

Contract technical monitor: Robert M. Bennett

16 Abstract

This document describes the design and usage of two main subprograms using direct methods to solve large linear complex systems, of the form $Ax = b$, whose coefficient matrices are too large to be stored in core. The first main subprogram is for systems whose coefficient matrices are of a particular sparse structure, namely, the matrix A can be written in the form $B + D$, where B is a block-banded system, and D has only a few columns of nonzeros. Key elements of the algorithms used in the subprograms include: the data structure, the strategy for preserving numerical stability, the adaptability of the algorithms for dense systems as well as for block-profile systems.

17 Key Words (Suggested by Author(s))	18 Distribution Statement
upper triangular matrix	
threshold pivoting block-profile matrix	
pivotal tolerance block-banded matrix	
rank	
Sherman-Morrison Updating Formula	

19 Security Classif. (of this report)	20 Security Classif. (of this page)	21 No. of Pages	22 Price*
Unclassified	Unclassified	39	